



Implementando Segurança em APIs RestFul

Ao desenvolvermos um projeto de software é comum considerarmos os requisitos funcionais e não funcionais do mesmo. Entretanto, ainda é pouco comum considerar, entre os aspectos não funcionais, toda a parte de segurança de software. Tal questão, normalmente, é pensada nas etapas finais do processo de desenvolvimento, fato esse que acaba gerando maiores custos (recursos, tempo, retrabalho, etc.) se comparado com a adoção de esforços, para planejar a segurança da aplicação nas etapas iniciais do projeto.

Antes de tratarmos da aplicação da segurança fazendo uso do Spring Boot, é importante reforçarmos alguns conceitos a respeito de tal tema. Falar de segurança na engenharia de software envolve tópicos como confiabilidade, tolerância a falhas, segurança da informação, controle de acesso, entre outros. Nesse sentido, confiabilidade e tolerância a falhas caminham juntos, estando ambos relacionados também a outro aspecto, o de maturidade do sistema e sua capacidade de se recuperar mediante a falhas. Já a segurança da informação está voltada para a proteção da aplicação, e seus dados, como um todo, envolvendo toda a preocupação com o controle de acesso, por exemplo.

Além dos aspectos mais amplos tratados acima, há alguns outros mais específicos, relacionados aos padrões de mercado/indústr



para implementação de segurança em aplicações. Nesse sentido, cabe destacar o Protocolo OAuth 2.0. Tal protocolo, mantido por um

grupo de trabalho (IETF OAuth Working Group) e que continua recebendo atualizações, especifica regras e fluxos para o processo de autenticação e controle de acesso.

Ao longo desse módulo daremos atenção justamente à segurança da informação – mais precisamente ao controle de acesso, seguindo os princípios do OAuth, suportado pelo módulo de segurança do Spring. Nesse sentido, veremos como proteger nossa aplicação, do ponto de vista da autenticação e do controle de acesso, a fim de impedir que pessoas ou sistemas não autorizados a acessem e também a fim de permitir que os seus diferentes recursos sejam acessíveis apenas a quem de direito. Enquanto a autenticação se preocupa com a questão "quem é você?" ou, melhor, "quem está tentando acessar a aplicação", o controle de acesso (também chamado de autorização) lida com a pergunta "o que você está autorizado a fazer?" ou "o que um determinado cliente, que está acessando a aplicação, está autorizado a fazer?".

Para responder a essas duas questões, o Spring possui uma arquitetura dedicada de segurança. Dentro da mesma é possível utilizar todos os recursos disponibilizados pelo framework, além de também ser possível customizá-los. Por se tratarem de assuntos bastante extensos, não abordaremos neste módulo, os detalhes da arquitetura em questão, empregada pelo Spring e providos pelo OAuth. Por outro lado, veremos como aplicar tanto a autenticação como a autorização em uma API Restful: protegeremos todo o acesso à API através de credenciais e configuraremos as rotas públicas e privadas, ou seja, quais endpoints poderão ser acessados sem a necessidade de credenciais, como os recursos de criação de usuário e autenticação, por exemplo. Além disso, estabeleceremos a base necessária para que, usando o modelo fornecido pelo Spring,

você possa definir o controle de acesso a cada endpoint de acordo com o perfil do usuário.

O primeiro passo para protegermos nossa API consiste na configuração da dependência – biblioteca “spring-boot-starter-security”. Em uma API pré-existente ou em uma nova API (utilize o site start.spring.io como ponto de partida, adicionando lá mesmo as dependências necessárias, incluindo a de segurança), a partir do gerenciador de dependências de sua preferência, insira os pacotes em questão. Abaixo, podemos ver a linha a ser inserida no “pom.xml” (em projetos Maven):

```xml
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
        <groupId>com.auth0</groupId>
        <artifactId>java-jwt</artifactId>
        <version>3.19.2</version>
</dependency>
```

Pom.xml: Inclusão das Dependências de Segurança

A partir desse momento nossa API contará com o módulo de segurança. Em termos práticos, isso implica na necessidade de fornecer um token de acesso para cada requisição feita para a mesma. Como ainda não configuramos o módulo de segurança e nem definimos o seu tipo, o próprio Spring gera o token em questão sempre que iniciarmos a API. Na figura 1, abaixo, podemos ver um exemplo desse token:

```
 :: Spring Boot ::                (v2.7.3)

2022-08-22 16:09:58.134  INFO 20900 --- [  restartedMain] b.c.d.p.seguranca.SegurancaApplication   : Starting SegurancaApplication using Java 11.0.2 on DESKTOP-SRHI8I3 wit
2022-08-22 16:09:58.135  INFO 20900 --- [  restartedMain] b.c.d.p.seguranca.SegurancaApplication   : No active profile set, falling back to 1 default profile: "default"
2022-08-22 16:09:58.147  INFO 20900 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-propertie
2022-08-22 16:09:58.147  INFO 20900 --- [  restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level
2022-08-22 16:09:58.625  INFO 20900 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-08-22 16:09:58.666  INFO 20900 --- [  restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 34 ms. Found 3 JPA reposit
2022-08-22 16:09:59.158  INFO 20900 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2022-08-22 16:09:59.166  INFO 20900 --- [  restartedMain] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-08-22 16:09:59.166  INFO 20900 --- [  restartedMain] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-08-22 16:09:59.237  INFO 20900 --- [  restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-08-22 16:09:59.237  INFO 20900 --- [  restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1070 ms
2022-08-22 16:09:59.258  INFO 20900 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-08-22 16:09:59.481  INFO 20900 --- [  restartedMain] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-08-22 16:09:59.491  INFO 20900 --- [  restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration    : H2 console available at '/h2-console'. Database available at 'jdbc:h2:
2022-08-22 16:09:59.627  INFO 20900 --- [  restartedMain] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-08-22 16:09:59.667  INFO 20900 --- [  restartedMain] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.10.Final
2022-08-22 16:09:59.792  INFO 20900 --- [  restartedMain] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-08-22 16:09:59.877  INFO 20900 --- [  restartedMain] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
2022-08-22 16:10:00.429  INFO 20900 --- [  restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.tra
2022-08-22 16:10:00.436  INFO 20900 --- [  restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-08-22 16:10:00.683  WARN 20900 --- [  restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database que
2022-08-22 16:10:00.918  WARN 20900 --- [  restartedMain] .s.s.UserDetailsServiceAutoConfiguration :

Using generated security password: 2bd90d44-7724-4f10-a29f-82205e9dff50

This generated password is for development use only. Your security configuration must be updated before running your application in production.

2022-08-22 16:10:01.027  INFO 20900 --- [  restartedMain] o.s.s.web.DefaultSecurityFilterChain     : Will secure any request with [org.springframework.security.web.sessior
2022-08-22 16:10:01.062  INFO 20900 --- [  restartedMain] o.s.b.d.a.OptionalLiveReloadServer       : LiveReload server is running on port 35729
2022-08-22 16:10:01.091  INFO 20900 --- [  restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2022-08-22 16:10:01.099  INFO 20900 --- [  restartedMain] b.c.d.p.seguranca.SegurancaApplication   : Started SegurancaApplication in 3.266 seconds (JVM running for 4.819)
```

Figura 1: Token de segurança gerado pelo Spring

Após a inserção da dependência, precisaremos incluir uma propriedade no arquivos de propriedades da API, o "application.properties". Tal dado será usado como chave para a criação do Token. Isso feito, o passo seguinte consiste na criação da classe de configuração do módulo de segurança. Nessa classe definiremos o comportamento do módulo, além dos endpoints/recursos que serão protegidos, e de outros detalhes. Em sua API, crie um novo package chamado "security". Nesse package, crie uma classe chamada "SecurityConfig". Essa classe estenderá a classe "WebSecurityConfigureAdapter" (que, conforme você perceberá, está depreciada. Contudo, manteremos a utilização da mesma, uma vez que a própria documentação do Spring, assim como a maioria dos tutoriais disponíveis na internet, continuam fazendo uso da mesma) e será composta por alguns métodos, fazendo override dos mesmos, a saber:

- configure: utilizado para a configuração das opções de segurança, como habilitação de componentes, definição das rotas protegidas e não protegidas, etc.;

- passwordencoder: utilizado para definir o encoder default para encriptação de senhas;

- authenticationmanager: utilizado no processo de autenticação do Spring Security.

Além dos métodos acima, outros poderão ser inseridos, posteriormente, na Classe, conforme requisitos específicos de segurança.

Um código inicial, e ainda não funcional, da Classe “SecurityConfig” pode ser visto abaixo.

```
//imports ....

//Define que essa classe e uma configuracao para o Spring
@Configuration
//Habilita a seguranca para a api
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired private
    UserRepository userRepo;
    @Autowired private
    JWTFilter filter;
    @Autowired private
    UserDetailsServiceImpl uds;

    //Metodo encarregado de configurar a seguranca da API
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable() // Desabilitando o csrf
                .httpBasic().disable() // Desabilitando a autenticacao por http basico(usuario e senha)
                .cors() // Habilitando o cors
                .and()
                .authorizeHttpRequests() // Autorizando requisicoes de entrada
                .antMatchers("/auth/**", "/h2-console/**").permitAll() // Autorizando requisicoes sem autenticacao para esse endpoint (nesse caso, path /auth/ e
uri´s subsequentes
                .antMatchers("/user/**").hasRole("USER") // Autorizando apenas usuarios com o perfil "USER" a utilizar esse endpoint (nesse caso, path /user/ e
uri´s subsequentes
                .anyRequest().authenticated() // Todas as requisicoes, exceto para as definidas acima, precisarao ser autenticadas
                .and()
                .userDetailsService(uds) // Setando o servico "user details" (do modulo Security do Spring) para essa implementacao customizada
                .exceptionHandling()
                    .authenticationEntryPoint(
                        // Rejeitando requisicoes nao autorizadas que chegam ate esse ponto.
                        //  Logo, isso significa que essa requisicao requer autenticacao e um JWT valido
                        (request, response, authException) ->
                            response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized")
                    )
                .and()
                .sessionManagement()
                .sessionCreationPolicy(SessionCreationPolicy.STATELESS); // Configurando a Sessao para que cada requisicao seja independente (stateless)

        // Adicionando o filtro JWT
        http.addFilterBefore(filter, UsernamePasswordAuthenticationFilter.class);
    }

    // Bean (tipo de Service) que sera responsavel por encriptar a senha
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    // Expondo o bean responsavel pelo gerenciamento do processo de autenticacao
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }
}
```

Sobre o código acima, cabem alguns comentários e explicações:

- Anotações @Configuration e @EnableWebSecurity: a primeira, dentro do padrão Spring, serve para informar que a classe em questão é uma classe de configuração e que, portanto, precisa ser “lida” no momento em que a API for executada. A segunda habilita o módulo WebSecurity, cujas configurações serão incluídas na classe;

- Anotação @Override: essa anotação não diz respeito, especificamente, ao módulo de segurança, mas ao framework como um todo. Através dela informamos que implementaremos uma sobrecarga de métodos;

- httpBasic().disable(): o Spring possui alguns recursos de segurança, como "loginPage" e "httpBasic". A configuração em questão desabilita a autenticação via "http básico" ;

- antMatchers() + permitAll() ou hasRole(): através da combinação dessas opções podemos configurar os endpoints que não requerem autenticação, os endpoints que controlarão o acesso de acordo com o perfil (role) do usuário, etc. Há várias combinações possíveis para essa opção;

- userDetailService: essa opção nos permite definir a classe responsável pela implementação do serviço "userDetail", pertencente ao Spring Security. Veja na documentação mais informações sobre o serviço em questão;

- http.addFilterBefore: essa opção indica que o filtro, definido como parâmetro na chamada do método, será aplicado antes das configurações de segurança. Há outras alternativas para essa configuração. Para saber mais, olhe a documentação do Spring Security;

- Anotação @Bean: também não restrita ao módulo de segurança, essa anotação informa, no framework, que o método em questão é um "Bean".

Sobre a implementação da Classe SecurityConfig, perceba que ela estende a WebSecurityConfigurerAdapter. Entretanto, essa ...na classe está depreciada atualmente. Ainda assim, apresento sua implementação, visto que a própria documentação oficial do Spring, assim como muitos tutoriais na internet, ainda a utilizam. Por outro lado, você poderá ver o código correspondente à classe SecurityConfig (assim como de toda a aplicação), sem utilização da WebSecurityConfigurerAdapter, no link mais abaixo.

Por último, precisamos deixar nossa Classe de configuração funcional, importando e instanciando os métodos de implementação do "userDetailService" e do filtro (usado no "addFilterBefore"). Ambos os códigos, assim como a Classe SecurityConfig e o "application.properties" se encontram disponíveis (devidamente comentados) no repositório abaixo. Você também encontrará uma coleção JSON para utilizar no Insomnia e, com isso, poderá testar a API.

Link:
https://github.com/FaculdadeDescomplica/pratica_integradora_tecnologias_disruptivas/tree/main/modulo5_**(Acesso em 28/09/2022)**

Versão da aplicação acima, sem utilização da classe depreciada WebSecurityConfigurerAdapter:

Link:
https://github.com/FaculdadeDescomplica/pratica_integradora_tecnologias_disruptivas/tree/main/modulo5b______**(Acesso em 28/09/2022)**

OBS: Recomendo, fortemente, que você analise todo o código, seguindo o fluxo desde o registro do usuário, depois pela autenticação e, por fim, pela utilização dos serviços disponibilizados a partir da verificação do token incluído nas requisições. O código, como já mencionado, está comentado e isso vai ajudá-lo em sua compreensão. Lembre-se de que todas as requisições, realizadas pelo Insomnia ou qualquer outro cliente, com exceção das requisições de registro e autenticação de usuário, precisarão conter o token (obtido na autenticação) em seu cabeçalho (header). O token é do tipo Bearer. No repositório acima, na coleção Insomnia, o método getAll Categoria possui um exemplo de configuração do token, inserido na aba "Auth", no Insomnia.

Nesse ponto, em nossa API, no que tange à configuração do módulo de segurança do Spring, já concluímos:

a) a inclusão da dependência necessária, no pom.xml;

b) a inclusão da chave para geração do Token no "application.properties";

c) a criação e configuração da classe de segurança e demais classes relacionadas, no package ".security";

d) a configuração, na classe de segurança, dos endpoints públicos – e, consequentemente, dos endpoints privados (ou seja, todos os endpoints que não são públicos são, portanto, privados e protegidos);

Para concluirmos o processo de implementação, veremos, a seguir, o fluxo de autenticação e de geração/validação de token, cujos códigos completos constam no repositório mencionado anteriormente.

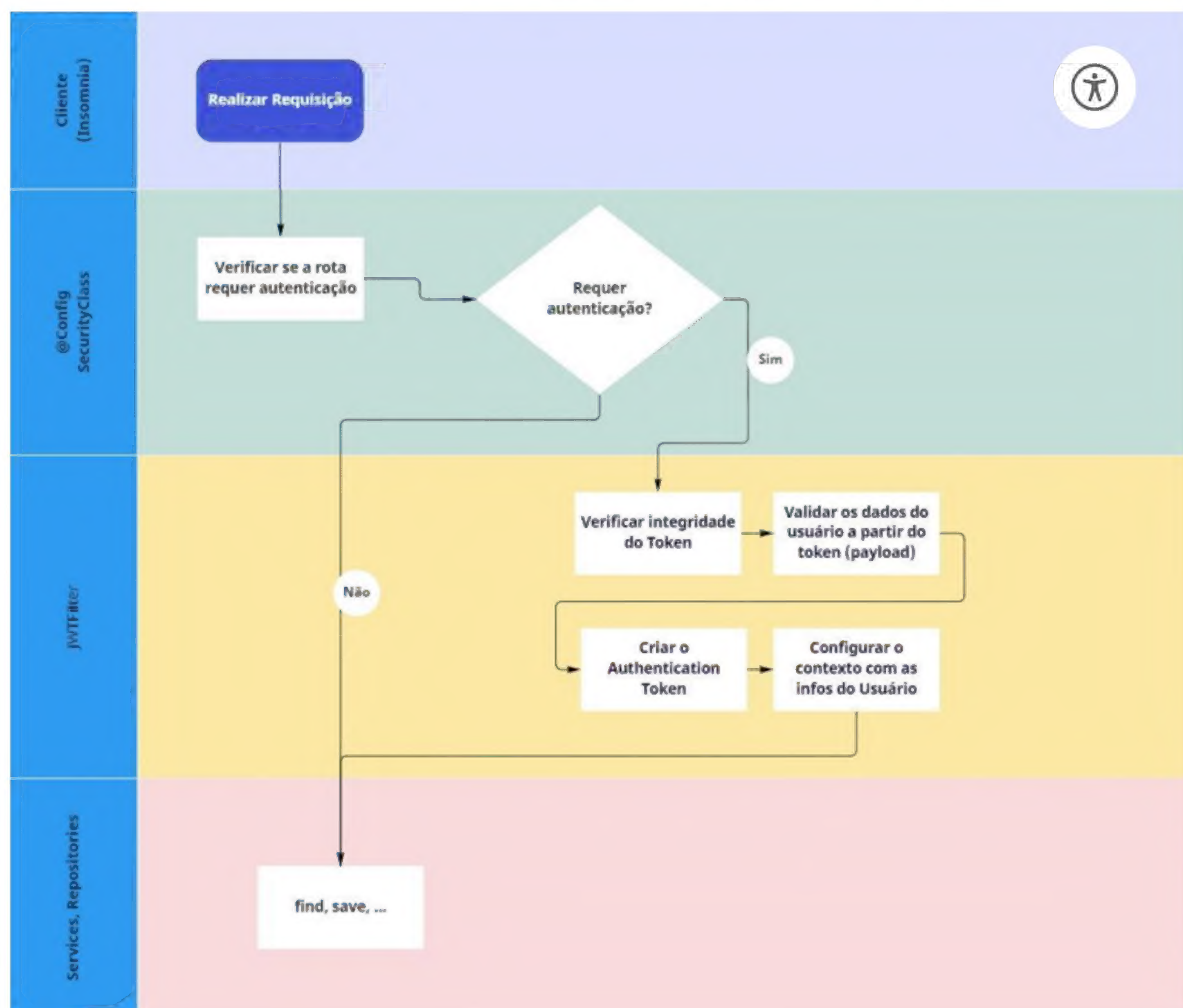


Figura 2: Fluxo de autenticação e autorização

No fluxo apresentado acima, e implementado no código disponibilizado, o controle de acesso e autenticação utilizam Token no formato JWT(JSON Web Token), que é um método de autenticação entre duas partes, por meio de um token, para requisições HTTP. O seu uso, por parte dos clientes da nossa API, tem seu início no acesso ao endpoint de login - que, conforme definimos na SecurityConfig, é um endpoint público. Após validação dos dados, o cliente recebe o token (no formato JWT). A partir desse momento, a cada nova requisição para os endpoints protegidos, ele precisará enviar, no cabeçalho (Header) de cada

requisição, o token como "Bearer token". Todo esse fluxo, e partes envolvidas no mesmo são apresentados na figura 2.

Com o detalhamento do fluxo de autenticação, chegamos ao final desse módulo, em que foram apresentados os conceitos de segurança em aplicações e o seu uso com o Spring Boot.

Atividade Extra

Se quiser conhecer em detalhes o módulo de segurança do Spring, leia os seguintes guias:

Links: https://spring.io/projects/spring-security ,

https://spring.io/guides/topicals/spring-security-architecture e

https://spring.io/guides/gs/securing-web/#:~:text=If%20Spring%20Security%20is%20on,Spring%20Security%20to%20the%20classpath **(Acesso em 28/09/2022)**

Referência Bibliográfica

SPILCĂ, Laurentiu. **Spring Security in Action.** Shelter Island, NY: Manning, 2020.



Atividade Prática Módulo 5

Título da Prática: Implementando segurança em uma API.

Objetivos: Configurar e utilizar o controle de segurança numa API Restful.

Materiais, Métodos e Ferramentas: IDE (Spring Tool Suite; Jetbrains Intellij; etc)

Atividade Prática

O controle de acesso e autenticação são etapas importantes na codificação de uma API. Em linhas gerais, configuramos nossa API para proteger, por padrão, todas as rotas, deixando públicas (sem controle de acesso) apenas alguns endpoints, como os de cadastro de usuário e autenticação. Com base no que foi discutido no Módulo 5, você deverá implementar o módulo de segurança em uma API. Para isso, siga o roteiro abaixo:

1. Inclua as dependências de segurança no pom.xml;

2. Inclua a chave (string) para geração do token no arquivo de propriedades;

3. Crie o package “security” e, nele, todas as classes de segurança, como a SecurityConfig e demais classes necessárias;

4. Crie a Entidade, Repositório, Serviço e Controle relacionados ao Usuário.

5. Teste a API utilizando o Insomnia ou o Postman.



Gabarito Atividade Prática

Como Gabarito para essa atividade, deverá ser utilizado o código disponível no repositório abaixo. Esse repositório é o mesmo visto no módulo e trata-se de um projeto funcional. Logo, a atividade não deverá consistir em "copia-e-cola", mas na prática do que foi visto através da codificação e respectivos testes, o que confirmará a conclusão, com sucesso, da atividade. Após isso, os códigos poderão ser comparados.

Link:
https://github.com/FaculdadeDescomplica/pratica_integradora_tecnologias_disruptivas/tree/main/modulo5 **(Acesso em 28/09/2022)**

Ir para exercício